Benchmarking Federated SPARQL Query Engines: Are Existing Testbeds Enough?

Gabriela Montoya[1], Maria-Esther Vidal[1], Oscar Corcho[2], Edna Ruckhaus[1], and Carlos Buil-Aranda[3]

[1] Universidad Simón Bolívar, Venezuela
{gmontoya, mvidal, ruckhaus}@ldc.usb.ve
[2] Ontology Engineering Group, Universidad Politécnica de Madrid, Spain
ocorcho@fi.upm.es
[3] Department of Computer Science, Pontificia Universidad Católica, Chile
cbuil@ing.puc.cl

Abstract. Testbeds proposed so far to evaluate, compare, and eventually improve SPARQL query federation systems have still some limitations. Some variables and configurations that may have an impact on the behavior of these systems (e.g., network latency, data partitioning and query properties) are not sufficiently defined; this affects the results and repeatability of independent evaluation studies, and hence the insights that can be obtained from them. In this paper we evaluate FedBench, the most comprehensive testbed up to now, and empirically probe the need of considering additional dimensions and variables. The evaluation has been conducted on three SPARQL query federation systems, and the analysis of these results has allowed to uncover properties of these systems that would normally be hidden with the original testbeds.

1 Introduction

The number of RDF datasets made publicly available through SPARQL endpoints has exploded in recent years. This fact, together with the potential added value that can be obtained from the combination of such distributed data sources, has motivated the development of systems that allow executing queries over federated SPARQL endpoints (e.g., SPARQL-DQP [2], Jena's ARQ[1], RDF::Query[2], ANAPSID [1], FedX [10], ADERIS [3]). Some systems use SPARQL 1.0 or ad-hoc extensions, while others rely on the query federation extensions that are being proposed as part of the upcoming SPARQL 1.1 specification [6].

In parallel to the development of federated SPARQL query evaluation systems, several testbeds have been created (e.g., as described in [2, 7, 8]), which complement those already used for single-endpoint query evaluation. The role of these testbeds is to allow evaluating and comparing the main characteristics of these systems, so as to provide enough information to improve them. Among

[1] http://jena.apache.org/
[2] http://search.cpan.org/~gwilliams/RDF-Query/

the features evaluated by these testbeds we can cite: *i*) functional requirements supported, *ii*) efficiency of the implementations with different configurations of datasets and with different types of queries, or *iii*) resilience to changes in the configurations of these systems and the underlying datasets. The most recent and complete testbed is FedBench [8], which proposes a variety of queries in different domains and with different characteristics, including star-shaped, chain-like and hybrid queries, and complex query forms using an adaptation of SP2Bench [9]. These testbeds are steps forward towards establishing a continuous benchmarking process of federated SPARQL query engines. However, they are still far from effectively supporting such benchmarking objectives. In fact, they do not specify completely or even consider some of the dependent and independent variables and configuration setups that characterize the types of problems to be tackled in federated SPARQL query processing, and that clearly affect the performance and quality of different solutions. This may lead to incorrect characterizations when these testbeds are used to select the most appropriate systems in a given scenario, or to decide the next steps in their development.

For example, testbeds like the one in [2] have limitations. First, queries are executed directly on live SPARQL endpoints; this means that experiments are not reproducible, as the load of endpoints and network latency varies over time. Second, queries were constructed for the data available in the selected endpoints at the time of generating the testbed, but the structure of these underlying RDF data sources changes, and may result in queries that are returning different answers or that do not return any answer at all. In cases like FedBench [8], the level of reproducibility is improved by using datasets that can be handled locally. However, as shown in Section 2, there are variables that are not yet considered in this benchmark (e.g., network latency, dataset configurations) and that are important in order to obtain more accurate and informative results.

The objective of this paper is to describe first the characteristics exhibited by these testbeds (mainly focusing on FedBench) and reflect on their current limitations. Additional variables and configuration setups (e.g., new queries, new configurations of network latency details, new dataset distribution parameters) are proposed in order to provide more accurate and well-informed overviews of the current status of each of the evaluated systems, so that the experiments to be executed can offer more accurate information about the behavior of the evaluated systems, and hence they can be used in continuous improvement processes. Finally, we describe briefly the results of our evaluation of this extended testbed using three different federated query engines: ARQ, ANAPSID, and FedX.

2 Some limitations of existing testbeds

There is no unique "one-size-fits-all" testbed to measure every characteristic needed by an application that requires some form of federated query processing [8]. However, regardless, existing testbeds can still be improved so that they can fulfill their role in continuous benchmarking processes.

We will first illustrate why we need to improve existing testbeds, particularly FedBench, by describing a scenario where the use of the testbed in its current form may lead to wrong decisions. We have executed the FedBench testbed with three systems (ANAPSID, ARQ, and FedX) on the three sets of queries proposed (Life Science, Cross Domain, and Linked Data) [4]. We have used different simulated configurations for network latencies and different data distributions of the datasets used in the experiments. As a result, we observe interesting results that suggest the need for improvements. For instance, for the Cross-Domain query `CD1`, all systems behave well in a perfect network (as shown in Table 1). However, their behavior changes dramatically when network latencies are considered. For instance, ARQ is not able to handle this query for medium-fast and fast networks, given the timeout considered; the time needed to execute the query in the case of FedX grows from 0.72 secs. (perfect network) to 2.23 secs. (fast network) and 16.93 secs. (medium-fast network); and for ANAPSID the results are similar for perfect and fast networks, and grows slower in medium-fast networks.

	Number of results			**Execution time (secs.) (first tuple)**			**Execution time (secs.) (all tuples)**		
Query Engine	**Medium**	**Fast**	**Perfect**	**Medium**	**Fast**	**Perfect**	**Medium**	**Fast**	**Perfect**
ANAPSID	61	61	61	0.98	0.17	0.16	0.98	0.17	0.16
FedX	61	61	61	16.93	2.23	0.72	16.93	2.23	0.72
ARQ	–	–	63	–	–	0.98	–	–	0.98

Table 1. Evaluation of FedBench query CD1-Number of results and execution time (secs.) under different network latency conditions. Timeout was set up to 30 minutes. Perfect Network (No Delays); Fast Network (Delays follow Gamma distribution ($\alpha = 1$, $\beta = 0.3$); Medium-Fast Network (Delays follow Gamma distribution ($\alpha = 3$, $\beta = 1.0$))

This is also the case for other FedBench queries (e.g., LD10, LD11, LS7, CD2), where different behaviors can be observed depending not only on network latency, but also on additional parameters, e.g., data distribution. What these examples show is that those parameters are also important when considering federated query processing approaches, and should be configured in a testbed, so as to provide sufficient information for decision makers to select the right tool for the type of problem being handled, or for tool developers to understand better the weaknesses of their systems and improve them accordingly, if possible.

Finally, there is also another aspect that is important when considering the quality of existing testbeds, and it is the fact that sometimes there are not sufficient explanations about the purpose of each of the parameters that can be configured. For example, in the case of FedBench there are several parameters that are considered when describing queries, as presented in [8], such as whether the query uses operators like conjunctions, unions, filters or optionals, modifiers like DISTINCT, LIMIT, OFFSET or ORDERBY, and structures like star-shaped queries, chains or hybrid cases. While this is quite a comprehensive set of features to characterize a SPARQL query, there are no clear reasons about why each of the 36 queries from the testbed are included. Only some ex-

amples are provided in [8], explaining that LS4 "includes a star-shaped group of triple patterns of drugs which is connected via owl:sameAs link to DBpedia drug entities", or that CD5 is a "chain-like query for finding film entities linked via owl:sameAs and restricted on genre and director". However, there are no explanations in the paper or in the corresponding benchmark website about the reasons for including each of them. Furthermore, there are parameters that are not adequately represented (e.g., common query operators like optionals and filters do not appear in cross domain or linked data queries), and characteristics that are not sufficiently discussed (e.g., the number of triple patterns in each basic graph pattern appearing in the query, the selectivity of each part of the query, etc.), which makes the testbed not complete enough.

In summary, while we acknowledge the importance of these testbeds in the state of the art of federated query processing evaluation, we can identify some of their shortcomings which we illustrate and describe in different scenarios.

3 Benchmark Design

In this section we describe some of the variables that have an impact on federated SPARQL query engines. There are two groups of variables: independent and dependent. Independent variables are those characteristics that need to be minimally specified in the benchmark in order to ensure that evaluation scenarios are replicable. Independent variables have been grouped into four dimensions: Query, Data, Platform, and Endpoint.

Dependent (or observed) variables are those characteristics that are normally influenced by independent variables, as described in Table 2, and that will be measured during the evaluation:

- *Endpoint Selection Time*. Elapsed time between query submission and the generation of the SPARQL 1.1 federated query annotated with the endpoints where sub-queries will be executed[3].
- *Execution Time*. This variable is in turn comprised of: *i*) *Time for the first tuple* or elapsed time between query submission and first answer, *ii*) *Time distribution* of the reception of query answers, and *iii*) *Total execution time*.
- *Answer Completeness*. Number of answers received in relation to the data available in the selected endpoints.

In the following sections we describe independent variables in more detail.

3.1 Query Dimension

This dimension groups variables that characterize the queries in terms of their structure, evaluation, and query language expressivity. Regarding the structure

[3] This variable is applicable only in cases where the system handles SPARQL 1.0 queries and no endpoints are specified in the query; hence, these queries have to be translated into SPARQL 1.1 or into an equivalent internal representation.

	Independent Variables	Observed Variables		
		Endpoint Selection Time	Execution Time	Answer Completeness
Query	query plan shape	✓	✓	✓
	# basic triple patterns	✓	✓	✓
	# instantiations and their position	✓	✓	
	join selectivity		✓	
	# intermediate results		✓	
	answer size		✓	
	usage of query language expressivity	✓	✓	
	# general predicates	✓	✓	✓
Data	dataset size		✓	
	data frequency distribution		✓	
	type of partitioning	✓	✓	✓
	data endpoint distribution	✓	✓	✓
Platform	cache on/off	✓	✓	
	RAM available	✓	✓	
	# processors	✓	✓	
Endpoint	#endpoints	✓	✓	✓
	endpoint type	✓	✓	
	relation graph/endpoint/instance		✓	✓
	network latency	✓	✓	✓
	initial delay	✓	✓	
	message size		✓	
	transfer distribution	✓	✓	✓
	answer size limit		✓	✓
	timeout		✓	✓

Table 2. Variables that impact the behavior of SPARQL federated engines

of the query, we focus on three main aspects: *i*) the query plan shape, *ii*) the number of basic triple patterns in the query, and *iii*) the instantiations of subject, object and/or predicates in the query.

Shape. Query plans may be star-shaped, chain-shaped or a combination of them, as described in [8]. In general, the shape of the input queries and of the query plans generated by the systems has an important impact on the three dependent variables identified in our evaluation (endpoint selection time, if applicable, execution time and answer completeness). The shape of the query plans will be in turn affected by the **number of basic triple patterns** in the query since this number will influence the final query shape. Query evaluation systems can apply different techniques when generating query plans for a specific type of input query, and this will normally yield different selection and execution times, and completeness results. For example, a query plan generator may or may not group together all graph patterns related to one endpoint.

Instantiations and their position in triple patterns. This is related to whether any of the elements of the triple patterns in the query (subject, object or predicate) are already instantiated, i.e., bounded to some URI. Together with **join selectivity**, instantiation has an important impact on the potential number of intermediate results that may be generated throughout query execution. For instance, the absence of instantiations (e.g., presence of variables) in the predicate position of a triple pattern may have an important impact in query execution time, because several endpoints may be able to provide answers for the pattern.

Answer size and **number of intermediate results**. If the number of answers or intermediate results involved in a query execution is large, it may

take a long time to transfer them across the network, and hence this may affect the query execution time.

Usage of query language expressivity. The use of specific SPARQL operators may affect the execution time and the completeness of the final result set. For example, the OPTIONAL operator is one of the most complex operators in SPARQL [5] and may add a good number of intermediate results, while the FILTER operator may restrict the intermediate results and answer size.

General predicates (e.g., `rdf:type`, `owl:sameAs`) are commonly used in SPARQL queries. However, as they normally appear in most datasets it is not always clear to which endpoint the corresponding subquery should be submitted, and this may have an impact in both endpoint selection and query execution time.

3.2 Data Dimension

We now describe the independent variables related to the characteristics of the RDF datasets that are being accessed. An RDF dataset can be defined in terms of its **size** and its **structural characteristics** like the number of subjects, predicates and objects, and the *in* and *out* degree of properties. These characteristics impact the number of triples that are transferred, and hence the total execution time. Additionally, they may affect the performance of the individual endpoints.

Partitioning and **data distribution** are two of the most important variables that need to be specified in the context of queries against federations of endpoints. Partitioning refers to the way that the RDF dataset is fragmented. Data distribution is the way partitions are allocated to the different endpoints. Data may be fully centralized, fully distributed, or somewhere in between. A dataset may be fragmented into disjunct partitions; the partitioning may be done horizontally, vertically or a combination of both. Horizontal partitioning fragments triples so that they may contain different properties. Vertical partitioning produces fragments which contain all the triples of at least one of the properties in the dataset. Horizontal partitioning impacts on the completeness of the answer whereas vertical partitioning affects the execution time. Partitions may be replicated in several endpoints, even in all of the endpoints, i.e., fully replicated, so that the availability of the system increases in case of endpoint failure or endpoint delay. Table 3 compares the behavior of ANAPSID and FedX with different configurations. The two engines behave similarly when there is one dataset per endpoint and in horizontal partitioning without replication. For vertical partitioning without replication, one engine is superior to the other. When partitioning with replication, one engine outperforms the other in vertical partitioning, and the inverse behavior occurs with horizontal partitioning.

Table 4 shows another example of the effect of data distribution on the query execution time, again for ANAPSID and FedX. We can observe that when there are multiple endpoints, results are similar, while with a network with no delay (perfect network) and all datasets in a single endpoint, one of the engines clearly outperforms the other in one order of magnitude.

Query Engine	Execution time First Tuple (secs.)	Execution time All Tuples (secs.)	Number of Results
One Dataset per Endpoint			
FedX	1.06	1.06	3
ANAPSID	1.08	1.28	3
Vertical Partitioning Without Replication			
FedX	0.69	0.69	3
ANAPSID	3.88	14.25	3
Horizontal Partitioning Without Replication			
FedX	0.72	0.72	3
ANAPSID	0.03	0.03	1
Vertical Partitioning With Replication			
FedX	0.85	0.85	14
ANAPSID	4.06	14.48	3
Horizontal Partitioning With Replication			
FedX	0.91	0.91	25
ANAPSID	0.06	0.06	1

Table 3. Impact of Data Partitioning and Distribution on FedBench query LD10 (Perfect Network). Vertical Partitioning: triples of predicates `skos:subject`, `owl:sameAs`, and `nytimes:latest_use` were stored in fragments. **Vertical Partitioning Without Replication**: three endpoints, each fragment in a different endpoint. **Vertical Partitioning With Replication**: corresponds to use four endpoints and store one of the three fragments in the four endpoints, another fragment in two endpoints, and the last fragment in one endpoint. Horizontal Partitioning: triples of the three predicates were partitioned in two fragments; each fragment has data to produce at least one answer. **Horizontal Partitioning Without Replication** two endpoints; one fragment in a different endpoint. **Horizontal Partitioning With Replicas**: four endpoints; one fragment is replicated in each endpoint, the other fragment in only one endpoint.

Results in Tables 3 and 4 support the claim that data partitioning, data distribution and network delays need to be explicitly configurable in testbeds.

3.3 Platform Dimension

The *Platform* dimension groups variables that are related to the computing infrastructure used in the evaluation. Here we include a minimum set of parameters, related to the system's cache, available RAM memory and number of processors, since this dimension may contain many more parameters that are relevant in this context, and that should anyway be explicitly specified in any evaluation setup when using this testbed.

Query Engine	Execution time First Tuple (secs.)	Execution time All Time (secs.)	Number of Results
Single Endpoint-All Databasets			
FedX	0.51	0.51	61
ANAPSID	0.045	0.046	61
Multiple Endpoints			
FedX	0.72	0.72	61
ANAPSID	0.17	0.17	61

Table 4. Impact of Data Distribution on FedBench query CD1 (Perfect Network). All Datasets in one endpoint versus datasets distributed in different endpoints

Turning the **cache management** function in the system together with the **available RAM** may affect greatly the query execution time. The meaning of dropping and warming up cache needs to be clearly specified as well as the number of iterations where an experiment is run in warm cache, and when cache contents are drooped off. In the context of federations of endpoints, information on endpoint capabilities may be stored in cache. The **number of processors** is also a relevant variable in the context of federated queries. If the infrastructure offers several processors, operators may parallelize their execution, and the execution time may be affected positively.

3.4 Endpoint Dimension

This dimension comprises variables that are related to the number and capabilities of the endpoints used in the testbed.

The first variable to be considered is the **number of SPARQL endpoints** where the query will be submitted and the **type of endpoints** that are used for the evaluation. The first variable affects all three observed variables, specially the result completeness because different endpoints may produce different answers. The **relationship between the number of instances, graphs and endpoints** of the systems used during the evaluation is also an important aspect that needs to be specified. Different configurations of these relationships may impact the three dependent variables.

There are several variables that have an important impact on the execution time, such as the **transfer distribution**, which is the time distribution of the transmission of packets by the endpoints, the **network latency**, which defines the delay in sending packets through the network, and the **initial endpoint delay**. An example of the impact of different network delays is illustrated in Table 5. Two queries from the Linked Data collection of FedBench were executed (LD10 and LD11). Note that ANAPSID and FedX behave similarly in LD10 when there is no delay; however, when delays are considered, FedX outperforms ANAPSID. On the other hand, in LD11 ANAPSID outperforms FedX when delays are present. In fact, ANAPSID is able to produce the first tuple after the same amount of time, independently of the delay.

Finally, SPARQL endpoints normally allow configuring a **limit on the answer size** of the queries and a **timeout**, so as to prevent users to query the entire dataset. This may generate empty result sets or incomplete results, particularly when endpoint sub-queries are complex.

4 Some Experimental Results

In this section we illustrate how the testbed extension can be used to better understand the behavior of some of the existing federated query engines. The extended testbed has been executed on three systems (ANAPSID, ARQ and FedX) with several configurations for the independent variables identified in

Query Engine	Query	Execution time First Tuple (secs.)	Execution time All Tuples (secs.)	Number of Results
Perfect Network				
ANAPSID	LD10	1.08	1.29	3
	LD11	0.06	0.09	376
FedX	LD10	1.06	1.06	3
	LD11	5.44	5.44	376
Fast Network				
ANAPSID	LD10	18.13	22.89	3
	LD11	0.06	2.80	376
FedX	LD10	3.45	3.45	3
	LD11	14.21	14.22	376
Medium Fast Network				
ANAPSID	LD10	191.78	241.58	3
	LD11	0.07	27.86	376
FedX	LD10	27.27	27.27	3
	LD11	108.93	108.93	376
Medium Slow Network				
ANAPSID	LD10	287.88	362.59	3
	LD11	0.05	41.74	376
FedX	LD10	41.42	41.42	3
	LD11	162.45	162.45	376
Slow Network				
ANAPSID	LD10	653.44	819.72	3
	LD11	0.09	92.52	376
FedX	LD10	87.19	87.19	3
	LD11	347.93	347.93	376

Table 5. Impact of Network latency on FedBench queries LD10 and LD11. Timeout was set up to 30 minutes and Message Size is 16KB. Perfect Network (No Delays); Fast Network (Delays follow Gamma distribution ($\alpha = 1$, $\beta = 0.3$); Medium-Fast (Delays follow Gamma distribution ($\alpha = 3$, $\beta = 1.0$); Medium-Slow (Delays follow Gamma distribution ($\alpha = 3$, $\beta = 1.5$); Slow (Delays follow Gamma distribution ($\alpha = 5$, $\beta = 2.0$))

Section 3. The complete result set generated by these executions can be browsed at the DEFENDER portal[4].

Now we will focus on one of the analyses that a system developer may be interested in, in the context of the continuous benchmarking process that we have referred to in this paper. That is, we are not analyzing the whole set of results obtained from the execution, but only a subset of it. Specifically, let's assume that we are interested in understanding the performance of the three evaluated systems under different data distributions in an ideal scenario, with no or negligible connection latency. Our hypothesis is that existing query engines are sensible to the way data is distributed along different endpoints, even when the network is perfect. Therefore, these results may be useful to validate that hypothesis and to understand whether a set of federated datasets for which we have the corresponding RDF dumps should be better stored in a single endpoint or in different endpoints to offer answers more efficiently. Based on the set of variables identified in our study, the following experimental setup is used:

Datasets and Query Benchmarks. We ran 36 queries against the FedBench dataset collections [8]: DBPedia, NY Times, Geonames, KEGG, ChEBI,

[4] `http://159.90.11.58/`

Drugbank, Jamendo, LinkedMDB, and SW Dog Food. These queries include 25 FedBench queries and eleven complex queries[5]. The latter are added to cover some of the missing elements in the former group of queries. They are comprised of between 6 and 48 triple patterns, and can be decomposed into up to 8 sub-queries; and they cover different SPARQL operators. Virtuoso[6] was used to implement endpoints, and the timeout was set up to 240 secs. or 71,000 tuples. Experiments were executed on a Linux Mint machine with an Intel Pentium Core 2 Duo E7500 2.93GHz 8GB RAM 1333MHz DDR3.

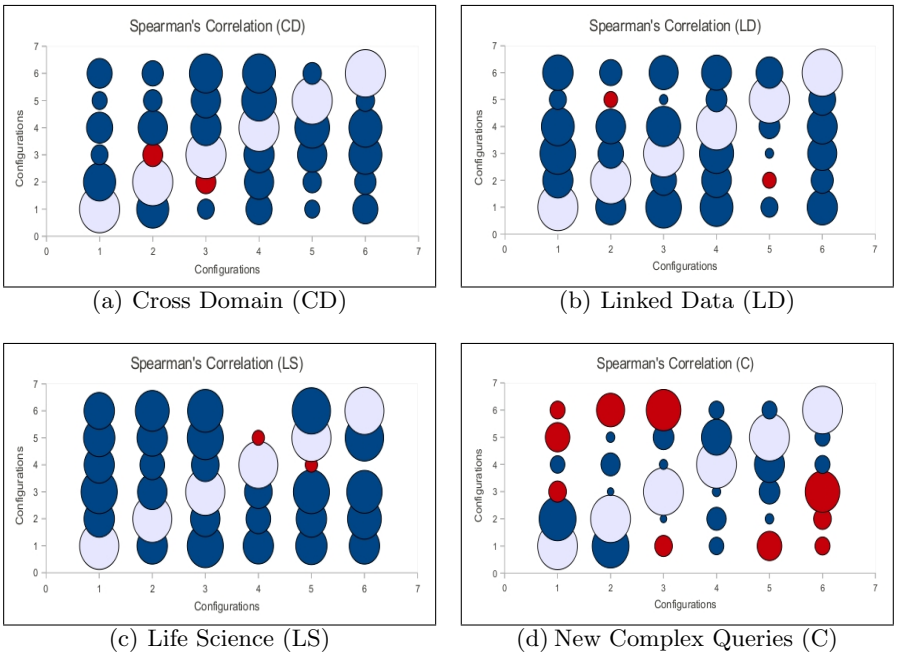
Network Latency. We configured a perfect network with no delays. The size of the message corresponded to 16KB.

Data Distribution. We considered two different distributions of the data: *i*) `Complete:` the FedBench collections were stored into a single graph and made accessible through one single SPARQL endpoint, and *ii*) `Federated`: the FedBench collections were stored in nine Virtuoso endpoints.

Therefore, we consider the queries in four groups and six configurations: **Configuration 1:** ANAPSID `Complete` Distribution, **Configuration 2:** ANAPSID `Federated` Distribution, **Configuration 3:** ARQ `Complete` Distribution, **Configuration 4:** ARQ `Federated` Distribution, **Configuration 5:** FedX `Complete` Distribution, **Configuration 6:** FedX `Federated` Distribution. In each configuration, the corresponding queries were ordered according to the total execution time consumed by the corresponding engines. For example, ANAPSID in a `Complete` Distribution, i.e., **Configuration 1**, the Cross-Domain queries were ordered as follows: CD2, CD3, CD4, CD5, CD1, CD7, and CD6. Queries of each configuration were compared using the Spearman's Rho correlation. A high positive value of correlation value between two configurations indicates that the corresponding engines had a similar behavior, i.e., the trends of execution time of the two engines are similar. Thus, when **Configuration 1** is compared to itself, the Spearman's Rho correlation reaches the highest value (1.0). On the other hand, a negative value indicates an inverse correlation; for example, this happened with Complex Queries to ARQ in a `Complete` Distribution (**Configuration 3**) when compared to FedX `Federated` Distribution (**Configuration 6**); its value is -0.757. Finally, a value of 0.0 represents that there is no correlation between the two configurations, e.g., for Life Science queries **Configuration 4** and **Configuration 6**. Figure 1 illustrates the results of this specific study (again, the data used for this study is available through the DEFENDER portal). White circles represent the highest value of correlation; red ones correspond to inverse correlations, while blue ones indicate a positive correlation. The size of the circles is proportional to the value of the correlation. Given a group of queries, a low value of correlation of one engine in two different distributions suggests that the distribution affects the engine behavior, e.g., FedX and ARQ in Complex Queries with different data distributions have correlation values of 0.143 and 0.045, respectively. Furthermore, the number of small blue circles between configurations of different data distributions of the same engine, indicate

[5] `http://www.ldc.usb.ve/~mvidal/FedBench/queries/ComplexQueries`

[6] `http://virtuoso.openlinksw.com/`



(a) Cross Domain (CD) (b) Linked Data (LD)

(c) Life Science (LS) (d) New Complex Queries (C)

Fig. 1. Spearman's Rho Correlation of Queries in three FedBench sets of queries (a) Cross-Domain (CD), (b) Life Science (LS), (c) Linked Data (LD) and (d) New Complex Queries. Six configurations: (1) ANAPSID `Complete` Distribution; (2)ANAPSID `Federated` Distribution; (3) ARQ `Complete` Distribution; (4) ARQ `Federated` Distribution; (5) FedX `Complete` Distribution; (6) FedX `Federated` Distribution. White circles correspond to correlation value of 1.0; blue circles indicate a positive correlation (Fig.1(d) points (3,4) and (5,6) correlation values 0.045 and 0.143, respectively); red circles indicate a negative correlation (Fig.1(d) points (2,6) and (6,3) correlation values -0.5 and -0.757, respectively). Circles' diameters indicate absolute correlation values.

that this parameter affects the behavior of the studied engine. Because there are several of these points in the Complex Queries plot, we can conclude that these two parameters (query complexity and data distribution) allow uncovering engines' behavior that could not be observed before. This illustrates the need for the extensions proposed in this paper.

5 Conclusion and Future Work

In this paper we have shown that there is a need to extend current federated SPARQL query testbeds with additional variables and configuration setups (e.g., data partitioning and distribution, network latency, and query complexity), so as to provide more accurate details of the behavior of existing engines, which can then be used to provide better comparisons and as input for improvement proposals. Taking those additional variables into account, we have extensively

evaluated three of the existing engines (ANAPSID, ARQ and FedX), and have made available those results for public consumption in the DEFENDER portal, which we plan to maintain up-to-date on a regular basis. We have also shown how the generated result dataset can be used to validate hypotheses about the systems' behavior.

Our future work plans will be focused on continuing with the evaluation of additional federated SPARQL query engines, and with the inclusion of additional parameters in the benchmark that may still be needed to provide more accurate and well-informed results.

6 Acknowledgements

This work has been funded by the project myBigData (TIN2010-17060), and DID-USB. We thank Maribel Acosta, Cosmin Basca, and Raúl García-Castro for fruitful discussions.

References

1. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: An adaptive query processing engine for SPARQL endpoints. In *Proceedings of the 10th International Semantic Web Conference*, volume 7031 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2011.
2. C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In *ESWC (2)*, pages 1–15, 2011.
3. S. J. Lynden, I. Kojima, A. Matono, and Y. Tanimura. ADERIS: An adaptive query processor for joining federated SPARQL endpoints. In *OTM Conferences (2)*, pages 808–817, 2011.
4. G. Montoya, M.-E. Vidal, and M. Acosta. DEFENDER: a DEcomposer for quEries against feDERations of endpoints. In *Extended Semantic Web Conference, ESWC Workshop and Demo 2012, LNCS*, 2012.
5. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *TODS*, 34(3), 2009.
6. E. Prud'hommeaux and C. Buil-Aranda. SPARQL 1.1 federated query, November 2011.
7. B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, pages 524–538, 2008.
8. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *International Semantic Web Conference (1)*, pages 585–600, 2011.
9. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2bench: A SPARQL performance benchmark. In *ICDT*, pages 4–33, 2010.
10. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference*, pages 601–616, 2011.